Application-Oblivious L7 Parsing Using Recurrent Neural Networks

Hao Li, Zhengda Bian, Peng Zhang, *Member, IEEE*, Zhun Sun, Chengchen Hu, *Member, IEEE*, Qiang Fu, *Member, IEEE*, Tian Pan, and Jia Lv

***Abstract*— Extracting fields from layer 7 protocols such as HTTP, known as L7 parsing, is the key to many critical network applications. However, existing L7 parsing techniques center around protocol specifications, thereby incurring large human efforts in specifying data format and high computational/memory costs that poorly scale with the explosive number of L7 protocols. To this end, this paper introduces a new framework named *content-based L7 parsing*, where the content instead of the format becomes the first class citizen. Under this framework, users only need to label what content they are interested in, and the parser learns an extraction model from the users' labeling behaviors. Since the parser is specification-independent, both the human effort and computational/memory costs can be dramatically reduced. To realize content-based L7 parsing, we propose REPLAY which builds on recurrent neural network (RNN) and addresses a series of technical challenges like large labeling overhead and slow parsing speed. We prototype REPLAY on GPUs, and show it can achieve a precision of 98% and a recall of 97%, with a throughput as high as 12Gbps for diverse extraction tasks.**

***Index Terms*— Application layer protocol, deep packet inspection, protocol parsing, recurrent neural networks (RNNs).**

I. Introduction

DEEP Packet Inspection (DPI) plays an irreplaceable role in improving the performance and security of computer networks. As more and more applications are encoding their data using flexible data formats (*e.g.*, XML/JSON), and embedding them deeply into the payload of application-layer protocols (*e.g.*, HTTP/HTTPS), DPI often needs to extract content from the payload of these application-layer protocols, commonly known as L7 parsing[1] [1], [2]. Even most L7 payloads are encrypted for private transmission on the Internet, service providers still heavily rely on L7 parsing inside their data centers where payloads are already decrypted, for realizing critical applications, *e.g.*, data loss prevention [3], [4], intrusion detection [5], big data analysis [6], and fine-grained load balancing [7]. Besides, the network operators utilize L7 parsing to troubleshoot their services by monitoring the traffic between systems, which are not encrypted in the first place, *e.g.*, NFV/SDN monitoring [8], [9], application performance management [10].

While L7 parsing has been heavily utilized in finer-grained and domain-specific applications, current L7 parsing techniques pose two serious issues.

Heavy human effort. Before extracting the content from L7 protocol payload, most L7 parsers require operators to specify the data format, *i.e.*, specification, used by each application. This requires heavy human effort even for a single application. For example, to analyze a remote file access behavior, the operators may have to explicitly learn the data format of NetWare Core Protocol, which has more than 400 individual request types [5], [11]. It can take even more effort if a protocol is proprietary, or constantly evolving. For example, Qosmos, a famous DPI solution vendor, has hired a team comprised of experienced engineers to ensure their 3000+ L7 specifications with 4500+ fields stay fresh [12].

Unscalable extraction overhead. Even with perfect data format, L7 parsing is still faced with a high online extraction overhead. First, sequentially matching each protocol is computationally expensive, making the parsing unacceptably slow, especially when there are a large number of protocols. On the other hand, merging the data formats of all protocols can achieve a higher parsing speed, but will cause the memory explosion problem [13]. For example, it is reported that the DFA that merges 794 signatures from Snort consumes 5.29GB memory [1]. Note that the above overhead, either in computation or in memory, depends on the number of specifications, rather than the traffic to parse. Thus, even though the majority of the traffic is irrespective, the L7 parsers will still incur the overhead by having to cover a complete set of specifications.

Many efforts have been made to mitigate the above problems. For the first problem, recent work attempts to learn the data formats from the raw traffic, *a.k.a.*, protocol reverse engineering [14]–[16]. However, preparing enough clean traffic for format learning is time-consuming, as operators often do not have the pre-knowledge of target protocols. Moreover, training on the singular protocol would lacks accuracy on multiple similar protocols, *e.g.*, reverse engineering on web applications

Manuscript received December 20, 2018; revised November 2, 2019 and March 31, 2020; accepted June 3, 2020; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor K. Argyraki. Date of publication June 24, 2020; date of current version October 15, 2020. This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB0801703 and Grant 2019YFB1802600, in part by the NSFC under Grant 61702407, Grant 61772412, Grant 61702049, and Grant 61902307, and in part by the Fundamental Research Funds for the Central Universities, InternetNZ, and MoE-CMCC Artifical Intelligence Project under Grant MCM20190701. The work of Peng Zhang was supported by the K. C. Wong Education Foundation. *(Corresponding author: Hao Li.)*

Hao Li, Zhengda Bian, and Peng Zhang are with the School of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China (e-mail: hao.li@xjtu.edu.cn).

Zhun Sun is with CAI, RIKEN, Tokyo 103-0027, Japan (e-mail: zhun.sun@riken.jp).

Chengchen Hu is with Xilinx Labs Asia Pacific, Singapore 486040 (e-mail: huc@ieee.org).

Qiang Fu is with the School of Engineering and Computer Science, Victoria University of Wellington, Wellington 6012, New Zealand (e-mail: qiang.fu@ecs.vuw.ac.nz).

Tian Pan is with the State Key Laboratory of Networking and Switching Technology, BUPT, Beijing 100876, China (e-mail: pan@bupt.edu.cn).

Jia Lv is with the Huawei Noah's Ark Lab, Hong Kong (e-mail: jia.lv@huawei.com).

Digital Object Identifier 10.1109/TNET.2020.3000430

[1] In this paper, we use the term L7 to collectively refer to all protocols that operate above layer 4, and define the task of extracting critical content from the payload of L7 protocols as L7 parsing.

may generate identical specifications, since they share the same format of the header (HTTP) and the content (HTML). For the second problem, previous work tries to minimize the combined automaton, but at the risk of lowering the parsing speed [17], [18].

We observe that the root cause of the above problems lies in the previous design principle of L7 parsers, *i.e.*, first generating the specifications, and then extracting the content accordingly. Such principle makes operators focus on "how" to extract, which varies in different specifications, and as a result, both the human efforts and computation/memory cost grow linearly or even exponentially with the number of specifications. This paper proposes a new design philosophy for L7 parsing that is fundamentally different from what its predecessors build on. In this design, the users first specify the content that they are interested in by labeling sampled traffic, based on which the parser automatically learns how to extract from incoming traffic. This design allows users to focus on "what" content to extract on a small data set without considering the data formats of the content. In this way, the human efforts and parsing overhead only depend on the content itself, instead of the specifications, leading to a more scalable performance.

Deep learning technique is an intuition to realize the above content-based approach, which however will face several challenges if implemented naively: (1) the labeling process can be even more costly than specification writing, since training an accurate model requires large amount of labels, and (2) the parsing speed tends to become the bottleneck due to the complex computations for each input byte.

In this paper, we propose REPLAY, a deep learning approach that realizes the content-based parsing, while addressing the above challenges. Specifically, REPLAY adopts the recurrent neural networks (RNN) as the footstone, and enhances it from several perspectives: (1) REPLAY combines semi-supervised learning and active learning schemes into RNN, so as to learn the extraction behaviors from only a *minor* amount of labeled data, and (2) REPLAY *selectively* parses the input that might be useful for extraction, which greatly accelerates the parsing due to the low value density of network traffic. Combining the above enhancements, we advocate that REPLAY is an application-oblivious L7 parser, which extracts fields for *all applications* by the labeled content at line rate, without any pre-knowledge of protocol specifications.

The main contributions of this work are as follows:

- We propose the framework of the content-based parser, whose application-oblivious property makes it possible to minimize human intervention while achieving high accuracy and minor overhead (Section III).
- We design an RNN equipped with semi-supervised learning, active learning and selective parsing techniques, so as to use a small amount of samples to train a model that is accurate and efficient in real time (Section IV).
- We implement REPLAY and evaluate it with diverse L7 traffic datasets. The experimental results show that REPLAY is able to achieve scalable performance of 12Gbps throughput with 98% precision and 97% recall in an intricate data environment (Section V and Section VI).

II. Current Practice and Problems

Generally speaking, existing L7 parsing can break down into two stages: off-line specification generation and on-line field extraction. In the following, we show how these two stages work, highlighting their inefficiency in both human labor and system performance.

```
From: alice@localhost
To: bob@localhost
Message-ID: 1413025098.42814.JavaMail.server
Subject: Editorial Decision Letter on ToN
MIME-Version: 1.0
Content-Type: multipart/mixed;
boundary="—-=_Part_42812_1784838041.1515575416501"

——=_Part_42812_1784838041.1515575416501
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: 7bit

It is a pleasure to accept your manuscript in its current form for
publication in the IEEE/ACM Transactions on Networking.
——=_Part_42812_1784838041.1515575416501–
.
250 2.0.0 Ok: queued as 7AB6219C15C9
```

Fig. 1. An SMTP message flow. The shadowed text are expected to be extracted.

A. Specification Generation

The goal of the specification generation stage is to obtain a specification on the data format for different application.[2] To achieve this, there are two options, *i.e.*, manually writing an accurate specification, or manually/automatically generating a coarse-grained specification. We will use a simple example to show the problems of these two options. As shown in Fig. 1, suppose we want to extract the `subject` and the `content` fields from emails sent through the SMTP protocol.

For the first option, we can manually write the format of SMTP protocol exactly following RFC 5321, which has 95 pages [19]. Due to the complexity of modern protocols, it can take days to write an accurate format for a single protocol [2]. Worse still, if a protocol is proprietary without public description, we need to manually analyze the format from the raw packets, which can take longer time. Considering there can be tens to thousands of L7 protocols, this manually specification writing process clearly requires prohibitive human efforts.

As another option, we can generate a coarse specification using regular expressions (RegEx). To extract the mail subject, we can write a RegEx `Subject: (.*)\r\n` to extract the subject field. This coarse specification can raise lots of false positives, as it can match arbitrary non-SMTP content containing the word "Subject", *e.g.*, an HTML page introducing the SMTP protocol [20]. For reducing such false positives, the current approaches go to another extreme: the operators write the RegEx-based specifications very carefully, to ensure that only the samples they have seen can match the specifications. This, in contrast, generates lots of false negatives. For example, l7filter as a famous RegEx-specification set, can achieve high precision (nearly 100%), but only with a recall of 70% [21]. Moreover, for mail content extraction, we need to locate and store the `boundary` field, which cannot be achieved by RegEx-based specification due to its low expressiveness.

Apart from manually writing, some methods learn RegEx by clustering similar strings from the clean traffic for a specific

[2]A specification corresponds to a certain application, which describes the format of the protocol of the application. As a result, we use "specification", "application" and "protocol" interchangeably.

application [14]–[16]. Such reverse-engineering methods lead to similar problems with handed-writing method, *i.e.*, potential false positive and low expressiveness. Besides, their performance heavily relies on the quality of input traffic. As a result, to obtain a sufficiently accurate specification, it takes much human effort to clean the input traffic beforehand, especially considering that the operator does not have any pre-knowledge of the protocol, *e.g.*, signature, port.

B. Field Extraction

Based on the specifications obtained from the last stage, the corresponding L7 parsers can be automatically generated [2], [5], [22], which aims to extract interested fields from real traffic. Considering multiple L7 protocols coexist, we need to extract fields based on multiple specifications simultaneously. There are roughly two ways to achieve this. One is rather straightforward: for each piece of content, we try each specification one-by-one, until there is a match. Obviously, the computation complexity of this method grows linearly with the number of specifications, and can result in prohibitive running time when there are a large number of specifications.

For the second method, we can combine all the specifications to achieve a scalable extracting speed. For example, we can merge multiple RegExes into a single one, so that the extracting time can be scalable with the number of specifications. However, this method can consume a large volume of memory due to the exponential increase of states. For example, it is reported that the DFA that merges 794 signatures needs 5.29GB memory [1], and merging only 38 specifications can take up to $5\times$ memory than the individual ones [13]. Recalling the potential large amount of specifications, the memory explosion could be a fatal issue of this mechanism.

In sum, the above overhead does not scale with the number of specifications, and always exists no matter what the inspected traffic or interested fields are. However, in real scenarios, the majority of the traffic would just be "noises" that do not interest operators. As a reference, the relative value density of the data generated in the Internet is only 6.25% [23]. Wasting so much computation/storage resources on the noisy data makes existing field extraction methods rather unscalable.

III. Application-Oblivious Parsing

A. Observations and Intuition

Our solution to the above problems of L7 parsing is based on the following key observations and intuitions.

Observations. After a close examination of the first stage of L7 parsing, *i.e.*, specification generation, we find that it naturally decomposes into two steps. First, the operator inspects the traffic samples and identifies the content that interests her, *i.e.*, the bytes she seeks to extract. Then, she could examine the context of the content, *e.g.*, keywords, separators around the content, and generalize it into an accurate specification. Our key observation is that the first step is relatively easy, but the second one is a highly demanding task.

1) *Identifying the interested content is easy.* The reason is threefold. First, the operator often focuses on a small portion of fields, *e.g.*, `Content-Type`, `Message-ID` in SMTP are irrelevant when analyzing the mail content. Second, the content to extract often has clear separators, *e.g.*, `\r\n` in HTTP, <> in XML. Third, most fields have semantic meanings and are easy to recognize, *e.g.*, `MAIL FROM` field in SMTP protocol. Thus, the interesting content can be easily identified even the operator does not have intimate knowledge of L7 protocols.
2) *Generalizing the specification is hard.* Recall that the goal of generalization is to summarize the data format specifying the way *all the content* is organized. As a result, the information of how the *interested content* is identified in the first step is of little help for this step: the identification only reveals the format patterns near the target content, while the rest part remains obscure to the operator without the aid of such content. If the operator summarizes the specification from the limited patterns, it would raise false positives as discussed in Section II-A; otherwise, she must refer to the protocol details for correct generalization.

Intuition. Based on the above observations, we identify the root cause of the hard specification generalization as the lack of the *uninterested information* in the sample traffic: they can rule out the false positives due to the true positive nature of the sample traffic, but are hard to summarize without the help of the intelligible content.

To this end, we have the following intuition to fulfill such missing part: we can let the operator remain focusing on the first step, *i.e.*, identifying the interested content, and let the parser itself generalizes an extraction model (rather than a concrete format specification) from the examined samples, by *learning* both their interested and uninterested contexts. This is essentially shifting the goal of generalization from the specification to the content, bypassing the difficulty of clarifying the uninterested chaos. As a result, the L7 parsing becomes application-oblivious, since we no longer need to care about data formats specified by different L7 applications. Moreover, as the parsing workload only depends on the content, the parsing speed can scale with the number of applications.

B. Content-Based L7 Parsing Framework

Based on the above intuition, we propose a new framework termed as content-based L7 parsing. Different from previous techniques, in our framework the content instead of the format becomes the first-class citizen. In a nutshell, operators explicitly indicate "what" content they are interested in by labeling it in sampled traffic, and based on the labeled traffic, the L7 parser learns "how" to extract them for real traffic. In the following, we will use an SMTP example to illustrate the workflow of content-based L7 parsing, which consists of two stages: *off-line training* and *on-line extracting*.

In the training stage, the operators are offered sample traffic, and should explicitly label the content that they want to extract. The labels are a sequence of tags, representing different *extraction behaviors* for the corresponding byte, *e.g.*, extracting or not, extracting as a key or a value. The parser takes the labeled traffic as input, and learns an *extraction model* that captures how the operators extract the content. As shown in the top of Fig. 2, the operator highlights the subject (*i.e.*, "accepted paper") in the SMTP payload. From such labeling behaviors, the parser learns the patterns in the complete context for content extraction, such as the separators (*e.g.*, colons and newlines) and keywords (*e.g.*, "From", "To", "250 OK"). Note that the training data contains flows that are with and without interested content, denoted by "positive" and

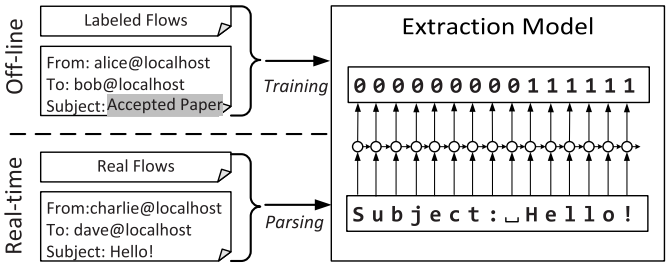


Fig. 2. A content-based parser learns the extraction behaviors from the labeled flows off-line, and extracts fields in real-time.

"negative" flows respectively, where the latter ones are learned to distinguish similar extraction behaviors.

In the extraction stage, the parser takes the byte sequence of each flow as the input, and outputs a *prediction*, *i.e.*, the extraction behaviors, based on the extraction model. As shown in Fig. 2, for an incoming real flow (left bottom), the parser outputs a sequence of extraction behaviors (right top), where 1 for extracting, 0 for not extracting. The final prediction is to extract the mail subject, *i.e.*, "Hello!" (right bottom).

There are two benefits for the content-based L7 parsing.

Little human effort. In the training stage, the operators only need to pick out the content they want to extract, without being disturbed by the complicated data format. Such application-oblivious nature results in two advantages on minimizing the human efforts: (1) operators are not required to have any pre-knowledge of protocol specifications, as shown in Section VI-C that even the inexperienced operators can generate the extraction model for 12 applications in a very short time; (2) each labeling process is independent from one another, so the whole labeling tasks can be conquered by multiple operators simultaneously, while in contrast, the specification writing is based on the knowledge gradually obtained through its process, which can hardly be parallelized.

Scaling with the number of applications. Instead of generating model for each application, the content-based parser extracts fields by a universal extraction model for all applications, which enables the scalability of the number of applications in real-time parsing: (1) the universal model performs exactly the same computations on all input sequences no matter how many applications are involved, and (2) the only storage cost of content-based parser is the parameters of the extraction model, which takes stable and minor memory.

In this way, we claim the content-based parsing framework is application-oblivious in both labeling and parsing.

C. Realizing L7 Parsing With RNN

In this paper, we employ a recurrent neural network (RNN) based approach to realize content-based L7 parsing. RNN is a type of artificial neural network that is able to process sequenced inputs, by the ability to perceive temporal information. RNN has been successfully employed in context-sensitive applications such as machine translation [24], speech recognition [25]. Specifically, given a sequence of d-dimensional inputs $\boldsymbol{X} = \{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^T$ and a specified time $t \leq T$, a RNN reads the input $\boldsymbol{x}_t$ as well as its previous hidden state $\boldsymbol{h}_{t-1}$, and updates its hidden state according to the mapping $\boldsymbol{h}_t = f_{\boldsymbol{\Theta}}(\boldsymbol{x}_t, \boldsymbol{h}_{t-1})$, then it outputs the current prediction by $\boldsymbol{o}_t = g_{\boldsymbol{\Theta}}(\boldsymbol{h}_t)$, where $\boldsymbol{h}$ and $\boldsymbol{o}$ are vectors with arbitrary dimensions, and f and g are functions parameterized by $\boldsymbol{\Theta}$. The goal of L7 parsing hence can be defined as obtaining the set $\mathcal{O}$ which contains all the possible sub-sequences $\hat{\boldsymbol{O}} \subset \boldsymbol{O}$ that make positive predictions, where $\boldsymbol{O} = \{\boldsymbol{o}_i\}_{i=1}^T$ is the output sequence.

However, the classical RNN has several problems that make it impractical to directly serve as the content-based parser.

Expensive labeling task. Even the content-based L7 parsing can potentially save human efforts in generating specifications, it is still necessary to label sampled traffic by operators. The classical RNN has to be trained in a fully supervised way [26], *i.e.*, all the training data should be labeled. As a result, the operators need to label a large amount of data to train an accurate extraction model.

Poor sample quality. The sample quality measures the degree to which the samples contribute to the training of extraction model. Samples with good quality should fully capture the contents to extract, which cannot be ensured by randomly picking from the unlabeled data sets. Recall the SMTP example in Fig. 1, if the mails in all positive samples have only one receiver, the extraction model may ignore the mails with multiple receivers in real traffic, since it has not seen such mails in the training stage.

Slow parsing speed. In classical RNN, each byte has to be computed through matrix multiplications and exponentiations, which heavily degrades the parsing speed. For example, the extraction model in Fig. 2 can only achieve ∼0.9Gbps throughput even with a high-end platform of two GPUs. The slow parsing speed greatly limits the application of classical RNN for real-time parsing in high-speed networks.

In the next section, we will show how REPLAY overcomes the above limitations by extending the classical RNN.

IV. Design of REPLAY

After setting a basis of REPLAY with RNN (Section IV-A), this section presents three techniques to enhance REPLAY, which respectively addresses the challenges listed in Section III-C: (1) the *semi-supervised learning scheme* to reduce the amount of required labeled data (Section IV-B); (2) the *active learning scheme* to select high-quality data for labeling (Section IV-C); and (3) the *selective parsing* to speed up the parsing process by dropping meaningless data (Section IV-D).

A. Setting a Basic RNN

Before introducing the enhancements, we need to first set a basic RNN structure for REPLAY. We note that the classical RNN lacks the complete understanding of the input: the context only includes states *before* the current input, *i.e.*, $\boldsymbol{h}_{t-1}$. For example, suppose we want to extract the `Method` field of HTTP protocol, *e.g.*, GET/POST. Since there is no other bytes before this field, classical RNN cannot collect enough context. As a result, we may end up *always* extracting the first few bytes of all traffic using RNN.

REPLAY adopts a typical *encoding-decoding structure* to address this problem [26], as shown in Fig. 3. First, the encoder reads the whole input sequence, and generalizes $\boldsymbol{s}_T$ as the complete understanding, where T is the length of the input. Then, the decoder takes $\boldsymbol{s}_T$ as its initial hidden state ($\boldsymbol{h}_0$), and performs typical recurrent computations to output the prediction. We present the architectures of the encoder and the decoder as follows.

Encoder. REPLAY constructs the encoder using the Gated Recurrent Unit (GRU), a gating mechanism that can adaptively

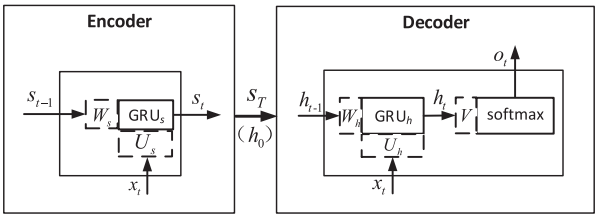


Fig. 3. The encoding-decoding RNN.

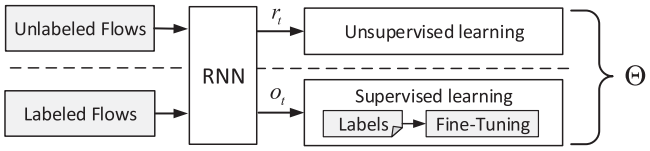


Fig. 4. Semi-supervised learning process.

capture the dependencies among different time steps, thereby achieving a more comprehensive generalization, especially for long input sequences [26]. As shown in the left part of Fig. 3, GRU_s calculates and passes on the hidden state $\boldsymbol{s}_t$, *i.e.*, $\boldsymbol{s}_t = \mathrm{GRU}_s(\boldsymbol{x}_t, \boldsymbol{s}_{t-1}; \boldsymbol{U}_s, \boldsymbol{W}_s)$, where $\boldsymbol{s}_0$ is initialized as a zero vector, $\boldsymbol{U}_s \in \boldsymbol{\Theta}$, and $\boldsymbol{W}_s \in \boldsymbol{\Theta}$ are weights of GRU_s. Here the bias parameters in GRU are omitted.

Decoder. The right part of Fig. 3 shows the decoder that outputs $\boldsymbol{o}_t$ as the prediction. Specifically, a hidden state $\boldsymbol{h}_t$ is calculated by GRU_h, *i.e.*, $\boldsymbol{h}_t = \mathrm{GRU}_h(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}; \boldsymbol{U}_h, \boldsymbol{W}_h)$, where $\boldsymbol{h}_0$ is initialized as the output of encoder, *i.e.*, $\boldsymbol{s}_T$, and $\boldsymbol{U}_h \in \boldsymbol{\Theta}$, $\boldsymbol{W}_h \in \boldsymbol{\Theta}$ are weights of GRU_h. Given $\boldsymbol{h}_t$, a softmax classifier is used to predict the label of $\boldsymbol{x}_t$, *i.e.*, $\boldsymbol{o}_t = \mathrm{softmax}(\boldsymbol{h}_t \boldsymbol{V})$, where $\boldsymbol{V} \in \boldsymbol{\Theta}$ is the weights of the classifier. Here we use the hierarchical softmax (H-softmax) [27] instead of the typical softmax function to ensure the scalability, which outputs an N-bit binary code to represent the selected extraction behavior (see details in Section V).

Basic learning process. Given the encoding-decoding structure, the learning phase of REPLAY is intuitively supervised: injecting the training input $\boldsymbol{x}$, for each time step t (each byte of the input), comparing the prediction $\boldsymbol{o}_t$ with the ground truth $\boldsymbol{y}_t$, and adjusting the computation weights accordingly. Specifically, the goal of training is to estimate the set of optimized global parameters $\boldsymbol{\Theta} = \arg\min_{\boldsymbol{\Theta}} J(\boldsymbol{\Theta})$. $J(\boldsymbol{\Theta}) = \frac{1}{T}\Sigma\mathbb{L}(\boldsymbol{o}_t, \boldsymbol{y}_t|\Theta)$ is the loss function to minimize, where $\mathbb{L}(\boldsymbol{o}_t, \boldsymbol{y}_t|\Theta)$ represents the error between $\boldsymbol{o}_t$ and $\boldsymbol{y}_t$.

However, due to the low value density, we have to deal with the class imbalance problem [28]: since only a few bytes of an input sequence are interesting to the operators, common loss function will encourage the extraction models to increase the accuracy by not extracting any byte(s). To overcome this problem, we introduce the proportion of the extraction behaviors into the loss function. Consider an extraction behavior i, its proportion for an input sequence will be:

$$prop(i) = \frac{\text{the number of bytes that fall into } i}{\text{total number of bytes}} \tag{1}$$

Next, we can add this factor into the loss function to balance the contribution from each extraction behavior, where the goal is to minimize the cross entropy (CE) between the ground truth $\boldsymbol{y}_t$ and the prediction $\boldsymbol{o}_t = p(\boldsymbol{y}_t|\boldsymbol{x}_{1:t}, \boldsymbol{\theta}_s, \boldsymbol{\theta}_h, \boldsymbol{\theta}_o)$:

$$J_L(\boldsymbol{\theta}_s, \boldsymbol{\theta}_h, \boldsymbol{\theta}_o) = \frac{1}{T}\sum_{t=1}^{T}\frac{\mathrm{CE}(\boldsymbol{y}_t, \boldsymbol{o}_t)}{prop(\boldsymbol{y}_t)} \tag{2}$$

where $\boldsymbol{\theta}_s = (\boldsymbol{U}_s, \boldsymbol{W}_s)$, $\boldsymbol{\theta}_h = (\boldsymbol{W}_h, \boldsymbol{U}_h)$.

We use Adam optimizer [29] and back-propagation through time (BPTT) [30] to optimize the parameters.

B. Reducing the Number of Required Labels

The supervised training of RNN requires lots of "expensive" labeled data [26]. On the other hand, the unlabeled data is very "cheap". Therefore, REPLAY adopts a semi-supervised learning (SSL) scheme to reduce the number of labels, by leveraging the unlabeled data. The novelty of this process is that REPLAY proposes a new autoencoder structure to realize SSL, which gains better understanding in L7 parsing scenario.

How SSL works. Fig. 4 shows the SSL process of REPLAY, which consists of two stages. In the first stage, SSL feeds the unlabeled data into RNN, and the decoder is expected to output extra information $\boldsymbol{r}_t$ for each input. REPLAY collects these outputs to perform an unsupervised optimization to obtain a rough $\boldsymbol{\Theta}$. In the second stage, based on the obtained $\boldsymbol{\Theta}$, SSL performs the typical supervised learning process as presented in Section IV-A. The warm-up unsupervised learning in the first stage can learn the inherent features of the unlabeled data, *e.g.*, the boundaries of different semantics (fields). Since the labeled data used in supervised phase also comes from the unlabeled data, those features are of great help to "complete" the patterns that are not described in the limited labeled samples, resulting in a model with better understanding.

The key of realizing the SSL process is the unsupervised learning phase. Specifically, in the phase we need to (1) extend the decoder to output an effective $\boldsymbol{r}_t$, and (2) design the training objectives to obtain an initial sensible $\boldsymbol{\Theta}$.

Extending decoder with autoencoders. Autoencoder is an essential component to realize SSL in conventional areas like question-answering system [31], [32]. The basic idea of autoencoder is to learn an approximation of the identical mapping function, which can yield a sequence $(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_T)$ that is similar to the input sequence $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$. This process is expected to generate a sensible initial $\boldsymbol{\Theta}$, which can provide more meaningful features for handling the labeled data than the randomly initialized weights.

The key that the autoencoder can reproduce the input sequence in previous natural language processing (NLP) scenarios [33], [34] is that it has a complete understanding from the encoder, *i.e.*, $\boldsymbol{S}_T$. However, T in L7 parsing scenario is much larger than it in NLP scenarios, *e.g.*, an HTTP response can have hundreds of kilobytes, while in NLP, the processing targets are mostly several sentences or paragraphs within 1KB. As a result, the information of $\boldsymbol{x}_{t+1}, \ldots, \boldsymbol{x}_T$ deduced from $\boldsymbol{S}_T$ would be "forgotten" after so many time steps, and can barely contribute to reproducing the original input $\boldsymbol{x}_t$. To this end, we design a new autoencoder structure for dealing with the long contexts in L7 parsing scenarios.

Fig. 5 presents the decoder extended with the newly designed antoencoders. The left part is its inner-encoding process to output $\boldsymbol{h}_t$, which acts as the decoder in Fig. 3, while the right part is the inner-decoding process to obtain $\boldsymbol{r}_t$ with the help of $\boldsymbol{h}_t$. Specifically, the inner-encoder will sequentially reads the input $\boldsymbol{x}_t$ and previous state $\boldsymbol{h}_{t-1}^{(fw)}$, and generates a forward hidden state $\boldsymbol{h}_t^{(fw)}$ at each time step t. After

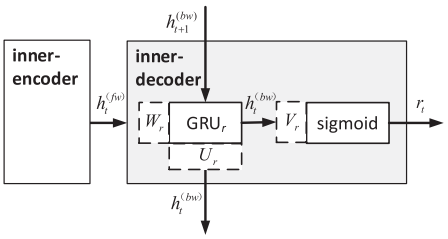


Fig. 5. Extended decoder with autoencoder.

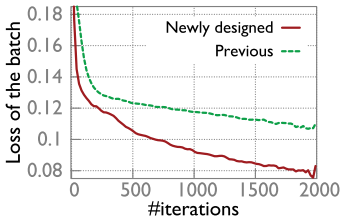


Fig. 6. Better understanding from the proposed autoencoder.

that, the inner-decoding GRU (GRU$_r$) in the inner-decoder reversely reads $\boldsymbol{h}_t^{(fw)}$ and previous state $\boldsymbol{h}_{t+1}^{(bw)}$, and generates a backward hidden state $\boldsymbol{h}_t^{(bw)}$ at the corresponding time step t. Next, a sigmoid function would process such states and output the $\boldsymbol{r}_t = \text{sigmoid}(\boldsymbol{h}_t^{(bw)}\boldsymbol{V}_r)$.

The intuition behind this process is as follows: if considering the propagation of the hidden state, we can roughly say that S_T is obtained by $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ in the encoder, and $\boldsymbol{h}_t^{(fw)}$ is obtained by S_T and $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t$ in the inner-encoder. Recall that $\boldsymbol{h}_{t+1}^{(bw)}$ is calculated using $\boldsymbol{h}_{t+2}^{(bw)}$ and $\boldsymbol{h}_{t+1}^{(fw)}$. The former embeds the information of $\boldsymbol{x}_{t+1}$ into $\boldsymbol{h}_{t+1}^{(bw)}$, while the later adds the information of $\boldsymbol{x}_{t+2}$ and $\boldsymbol{h}_{t+3}^{(bw)}$. In sum, $\boldsymbol{h}_{t+1}^{(bw)}$ contains the information of $\boldsymbol{x}_{t+1}, \ldots, \boldsymbol{x}_T$. Intuitively, $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t$ and $\boldsymbol{x}_{t+1}, \ldots, \boldsymbol{x}_T$ can decide the encoding of the separator, *i.e.*, $\boldsymbol{x}_t$, while $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T$ can ensure that the above two sub-sequences are from the same input. Therefore, the combination of $\boldsymbol{h}_t^{(fw)}$ and $\boldsymbol{h}_{t+1}^{(bw)}$ could generate an informative $\boldsymbol{h}_t^{(bw)}$ to obtain an accurate $\boldsymbol{r}_t$ for reproducing $\boldsymbol{x}_t$.

In the literature, the bidirectional RNN [35] also employs the backward hidden state to better understand the current context. The major difference is that the bidirectional RNN directly uses the forward and backward information to output the final prediction, while our design leverages those information to reproduce the input. In other words, our design combines the merits of autoencoder and bidirectional RNN, which can provide better understanding and faster convergence.

Unsupervised training objectives. Recall the grouped parameters of $\boldsymbol{\Theta}$, the unsupervised training phase would obtain an initial parameters of the encoder and inner-encoder, *i.e.*, $\boldsymbol{\theta}_s$ and $\boldsymbol{\theta}_h$. We further define a new parameter set $\boldsymbol{\theta}_r : \{\boldsymbol{W}_r, \boldsymbol{U}_r\}$. Then the goal of unsupervised training is to minimize the difference between the current input $\boldsymbol{x}_t$ and $\boldsymbol{r}_t$ over unlabeled samples, by calculating the mean squared error (MSE):

$$J_U(\boldsymbol{\theta}_s, \boldsymbol{\theta}_h, \boldsymbol{\theta}_r) = \frac{1}{T}\sum_{t=1}^{T} \text{MSE}(\boldsymbol{r}_t, \boldsymbol{x}_t) \tag{3}$$

Fig. 6 compares the convergence effect of previous and newly designed autoencoder on the same data set, where we use the training loss, *i.e.*, the deviation between the ground truth and the prediction, to measure the effectiveness of the model. It shows that the new autoencoder can decrease $\sim$30% training loss. Note that the previous autoencoder reproduces the input sequence by iteratively predicting the *next input*, while our design is reproducing the *current input*. However, as they both reproduce the complete input sequence, the training loss in Fig. 6 is calculated based on the same input (original sequence) and the same output (the reproduced sequence), which is still a fair comparison.

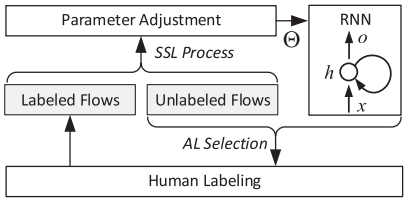


Fig. 7. Starting with a minor amount of labeled flows, BMAL selects informative unlabeled samples for labeling manually.

C. Improving Sample Quality

Though SSL largely reduces the number of required labeled flows, the sample quality remains a concern. Specifically, we cannot ensure the diversity of the labeled flows used in SSL, so the learned model may not completely reflect the extraction behaviors. The other problem related to the sample quality is the data amount to be checked: to obtain enough good-quality flows, the operator needs to check many more unlabeled flows, which heavily burdens the labeling work.

REPLAY applies the batch mode active learning (BMAL) scheme as a complementary to SSL, which iteratively selects informative unlabeled data for human labeling [36], [37]. The novelty of this process is that REPLAY propose an algorithm for identifying the high-quality samples with H-softmax.

How BMAL works. Fig. 7 depicts the learning process combining SSL and BMAL. At each iteration, BMAL selects a batch of samples from the unlabeled flows with the largest uncertainty for labeling manually, and SSL would be retrained with the additional labeled samples. The learning process ends when the trained model can ensure high accuracy. With such high-quality samples, the accuracy of the extraction model can be ensured with smaller amount of labeled flows. Besides, through this process, the operator need not manually or randomly check the unlabeled flows, but focuses on the most informative ones provided by BMAL.

The key of BMAL is, it identifies the samples with largest uncertainty (lowest confidence) are more informative for describing the boundaries of the extraction behaviors. In the following, we will first propose the algorithm that calculates the confidence, and then present the workflow of the BMAL.

Selecting algorithm. Recall the H-softmax function outputs an N-dimensional vector $\boldsymbol{o}_t$ to represent the final extraction behavior for each input $\boldsymbol{x}_t$, where each dimension has a probability $\boldsymbol{o}_{t,i}$. Then, for any possible extraction behavior $j \in \{1, \ldots, 2^N\}$ and its N-bit binary representation $\boldsymbol{b}_j$, we can calculate the similarity between $\boldsymbol{o}_t$ and $\boldsymbol{b}_j$ by each bit, as $g(\boldsymbol{o}_{t,i}, j) = \boldsymbol{b}_{j,i}\boldsymbol{o}_{t,i} + (1 - \boldsymbol{b}_{j,i})(1 - \boldsymbol{o}_{t,i})$. The probability of selecting j as the final decision for $\boldsymbol{o}_t$ is calculated as:

$$p(j|t) = \prod_{i=1}^{N} g(\boldsymbol{o}_{t,i}, j)) \tag{4}$$

$j_s = \arg\max p(j|t)$ is selected as the final extraction. It is obvious that a low $p(j_s|t)$ means that the model cannot

well distinguish all the extraction behaviors, *i.e.*, with large uncertainty.

Finally, we take the variance $C_d = \frac{1}{T_d}\sum_{t=1}^{T_d} p(j_s|t)^2$ as the confidence of the sample d, where T_d is the length of d.

D. Speeding Up Extraction

To obtain an accurate prediction over each input, RNN has to check the complete input sequence, which heavily lowers the parsing speed, considering the complex computation in each time step. Previous L7 approaches face the similar challenge, and they address it by skipping the useless bytes through parsing, *a.k.a.*, selective parsing (SP). Their principle is to skip the pure content that is irrelevant to resolving the protocol format. For example, when parsing a HTTP response with `Content-Length` field, the payload of this response can be directly skipped by counting the length. However, content-based parser cannot simply apply this technique for acceleration due to the absence of specifications.

REPLAY designs a novel scheme to effectively realize SP in RNN. Specifically, at each time step, REPLAY checks whether the rest input contains interested content, and drops the whole input if the answer is no. Since the real traffic is noise-dominating, REPLAY can expect a much higher throughput by early dropping the long tails of the meaningless data.

Design of stop trigger. Recall the encoder of RNN that reads the whole input, we can reuse this component to output the stop decision for input sequence, besides $\boldsymbol{s}_t$, since this process can capture the comprehensive understanding of the input. To be specific, another non-recurrent layer (sigmoid) is added into encoder to read $\boldsymbol{s}_t$ and output $z_t = \lfloor z_{t-1}\text{sigmoid}(\boldsymbol{s}_t\boldsymbol{V}_z)\rceil$, where z_t is the probability of continuing parsing, named "stop trigger". That is, once z_{t-1} is small than 0.5, all of the next z_t will be zero, so that the stop decision can be propagated.

We then compute a new hidden state at time step t, $\widehat{\boldsymbol{s}}_t = \boldsymbol{s}_{t-1}(1 - z_{t-1}) + \boldsymbol{s}_t z_{t-1}$. That is, if we decide to stop the parsing, *i.e.*, $z_{t-1} = 0$, $\widehat{\boldsymbol{s}}_t$ will be the same with $\boldsymbol{s}_{t-1}$.

In the training stage, since we have to perform a back derivative for tuning the parameter in the encoder, the computation would continue with this same $\widehat{\boldsymbol{s}}_t$. Recalling the rounding update of z_t, this ensures the final hidden state of the encoder, *i.e.*, $\widehat{\boldsymbol{S}}_T$, is the same with $\boldsymbol{s}_{\hat{t}}$, where $\hat{t}$ is the time step that pulls the stop trigger. As a result, the decoder in the training stage will not try to understand the dropped bytes.

While in the testing stage, the encoder will directly truncate the input if finding $z_t = 0$, and the decoder will only process the truncated input, which is the key to save the computation.

Training stop trigger. The stop trigger is trained after the SSL process. The only parameter to be trained is the weight for the sigmoid function, *i.e.*, $\boldsymbol{\theta}_z = \{\boldsymbol{V}_z \in \boldsymbol{\Theta}\}$, since $\boldsymbol{U}_s$ in the encoder has been well adjusted. We tune $\boldsymbol{\theta}_z$ in a supervised way by again using the labeled flows obtained in BMAL process. We then use SGD and BPTT to fine tune z_t by comparing $\widehat{\boldsymbol{o}}_t$ with the ground truth $\boldsymbol{y}_t$, to minimize the cross entropy $J_Z(\boldsymbol{\theta}_z)$ between $\boldsymbol{y}_t$ and the prediction $\widehat{\boldsymbol{o}}_t = p(\boldsymbol{y}_t|\boldsymbol{x}_{1:t}, \boldsymbol{\theta}_z)$:

$$J_Z(\boldsymbol{\theta}_z) = \frac{1}{T}\sum_{t=1}^{T}\frac{\text{CE}(\boldsymbol{y}_t, \widehat{\boldsymbol{o}}_t)}{prop(\boldsymbol{y}_t)} \tag{5}$$

We note that some RNNs (*e.g.*, word2vec [38]) employ the "end of sentence" (EOS) token to "stop" analyzing the sentences, which is a quite different concept with SP. The key is that in the NLP scenario, the EOS token can be *manually specified* in the text, *e.g.*, the period or the line break. While in L7 parsing, we do not have such natural terminals, instead, REPLAY *learns* the proper location to stop the parsing.

V. Implementation

We implement REPLAY with TensorFlow r1.8 [39] with ∼1800 lines of Python code, including all training and testing modules. Here we highlight several key designs.

Embedding. We encode each byte into an 8-dimensional vector with the value of its binary ASCII code before feeding it into REPLAY. Compared with the commonly used one-hot embedding, this embedding approach trades off the accuracy for efficiency. Ideally, the memory size of embedding can be reduced by $\frac{31}{32}$ with ASCII-based embedding compared with the one-hot embedding, as row size of RNN is reduced from 256 to 8. That is, given the limited memory, REPLAY can process much more neural cells in parallel with ASCII-based embedding to accelerate the inference. The results in Table III show that REPLAY with ASCII-based embedding is three times faster than it with the one-hot embedding, at the cost of only 1.8% degradation of accuracy.

H-Softmax. The typical softmax function outputs $\boldsymbol{o}_t$ as an N-dimensional vector, where each dimension represents the probability of an extraction behavior for $\boldsymbol{x}_t$. That is, the size of $\boldsymbol{o}_t$ and other weight vectors in $\boldsymbol{\Theta}$ will linearly increase with the number of extraction behaviors, which heavily slows down the computation in RNN. Instead, REPLAY employs H-softmax function, which does not output the probabilities of all extraction behaviors. Instead, it outputs an N-dimensional vector, where each dimension indicates the probability of selecting 0 and 1: if the probability is less than 0.5, it should be 0, otherwise it should be 1. In the end it will result in a binary encoding with length N, which represents a certain extraction behavior. In other words, the N-dimensional vector can classify 2^N extraction behaviors. In our prototype, REPLAY can classify 2^{16} extraction behaviors with a 16-dimensional vector for each $\boldsymbol{o}_t$, which is plenty enough to ensure the performance scalability with the number of applications.

Extraction post-processing. The content-based parser outputs extraction behaviors for each byte, so it is possible the final extracted content is not continuous as a single field, making it hard to be directly used in real scenarios like SDN control system. To this end, we propose a simple heuristic to post-process the extractions for producing a complete yet still accurate extraction result. Specifically, for each extraction behavior, if the extracted content is continuous, we directly use this result. Otherwise, we use the longest continuous extracted content as the baseline, and calculate the distance between it and other extracted content. If the distance is within a threshold, REPLAY will identify the gap text as the false negative, and merge the baseline and the gap text as well as the separated extracted content to compose the final result. Otherwise, REPLAY will identify the separated extracted content as the false positive and directly drop them. Experiment in Section VI-E shows that this heuristic works well for our data set, which can achieve even higher accuracy, because some errors are fixed through this heuristic.

Hyperparameters. When training with SSL, all the weights in Θ are initialized by sampling from a truncated normal distribution with standard deviation of 0.01, and all the bias parameters are initialized to zeros. To avoid the over-fitting

problem, we employ a dropout wrapper [40], [41] with a keep probability of 0.5 and L2 regularization with scalar of 0.01. We set the learning rate initially to be 0.001, and dynamically reduce it when the loss stops decreasing during training. We end the training if the loss on the whole epoch converges to a stable value, *e.g.*, not decreasing in continuous 10 iterations.

GPU platform. GPU platform is an intuitive choice for implementing a computation-intensive system like REPLAY. One major concern of this heterogeneous processor is how to well use the large number of processing units without performance penalty brought by frequent I/O with CPU. Our implementation builds multiple pipelines to asynchronously feed training/testing flows from CPU memory to GPU, and the GPU processing units will greedily read data from these pipelines to maximize the parallel processing ability.

VI. Evaluation

In this section, we experimentally evaluate REPLAY. Our experiments cover a series of realistic data set with many related approaches, which answer the following questions:

(1) Can REPLAY achieve high throughput and high accuracy at the same time? We find that REPLAY can reach over 10Gbps throughput while ensuring ∼97% precision and recall on the mixed data sets (Section VI-B).
(2) Is REPLAY cost-effective to learn an extraction model? We find that even an amateur operator can handle 12 applications with various extraction requirements in 3 hours with the help of REPLAY (Section VI-C).
(3) Does the model learned by REPLAY generalize to the new extraction cases? Our results suggest that REPLAY can maintain its high accuracy, even tested on a different dataset or for more complex data formats (Section VI-D).
(4) How sensitive is REPLAY to various factors? Our results show that the performance of REPLAY is largely unaffected by these parameters, *e.g.*, 20 initial labeled flows are sufficient for high accuracy (Section VI-E).

TABLE I
THE TRAFFIC OF COLLECTED APPLICATIONS

Application	Collected from/by	Extracting field(s)	Extract rate
IMDb [42] (HTML)	web spider	movie name, storyline	0.02%
Asahi [43] (HTML)	web spider	headline, news content	10.40%
RSS (XML)	XmlGenerator [44]	channel title	1.15%
SOAP (XML)	SoapUI [45]	header	1.38%
Weibo (JSON)	Weibo API	tweet text	18.20%
YouTube (JSON)	YouTube API	video name	5.60%
HTTP	real traffic	all headers	45.75%
QQ	real traffic	QQ number	0.28%
DNS query	real traffic	domain name	33.68%
FTP	real traffic	file name	1.33%
SMTP	SMTP sampler from jmeter [46]	sender,receiver subject,content	71.50%
MySQL	random query generator [47]	SQL query	30.50%

TABLE II
THE GROUPED DATA SETS

Set	Apps.	Extract rate	Set	Apps.	Extract rate
$A1$	{SMTP}	71.50%	$A7$	$A6$+{Youtube}	3.17%
$A2$	$A1$+{Weibo}	60.85%	$A8$	$A7$+{HTTP}	4.06%
$A3$	$A2$+{SOAP}	49.73%	$A9$	$A8$+{QQ}	4.06%
$A4$	$A3$+{IMDb}	2.09%	$A10$	$A9$+{DNS}	4.07%
$A5$	$A4$+{Asahi}	4.14%	$A11$	$A10$+{FTP}	4.05%
$A6$	$A5$+{RSS}	3.59%	$A12$	$A11$+{MySQL}	4.09%

A. Methodology

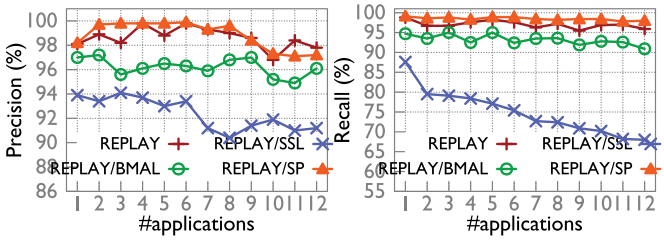
Data set. To evaluate the application-independence of REPLAY, we collect L7 traffic from 12 applications. Each traffic consists of 5,000 samples, where 3,000 of them are used as the training set, and the rest forms the testing set. We involve not only the typical L7 protocols, *e.g.*, HTTP, SMTP, but also user-defined protocols embedded in L7 payload, *e.g.*, YouTube video information based on JSON, RSS based on XML. We obtain above traffic from the real world or by widely used traffic sampler/tester for a realistic evaluation. Table I depicts the details of the collected traffic, where "extract rate" represents the average proportion of the extracting bytes from the flows. The "real traffic" in the table is collected from a campus network, which is anonymized before given to us. Note that all the samples in the data set have been automatically labeled according to manually generated specifications, while for the testing samples, such gold labels are only used when verifying the correctness of extraction.

To simplify the experiments, we randomly group all applications into 12 sets as shown in Table II. We also prepare a pure noise data set without any interested information.

Ethics statement. The operators who capture the traces from the real users anonymized the traces by the following steps: (1) reassembling the L4 payload (*i.e.*, L7 data) and removing all L2-L4 headers; (2) modifying the privacy-related fields (*e.g.*, QQ number) through an irreversible hash function. We only have the rights to access the anonymized traces. We also discuss other privacy issues in Section VII.

Involved approaches. For comparison, we introduce several approaches into the evaluations alongside with REPLAY. To evaluate the impact of the proposed three techniques, *i.e.*, SSL, BMAL, SP, we remove each of them from REPLAY to implement three sub-optimal approaches, denoted as REPLAY/SSL, REPLAY/BMAL, and REPLAY/SP, respectively. As for specification-based approach, we use the binaries from COPY [13], which can be configured as a sequential- or combined-specification parser, referred as sequential and combined parser respectively.

Experimental settings. For training experiments, we mix the training sets of all 12 applications, and combine them with pure noises to make an unlabeled training set that consists of 100K flows. For testing experiments, we select 2000 samples in total from the target application testing sets, and combine them with other testing sets to keep the noise rate (the volume of the uninterested flows divided by the total traffic volume) to be 80%. For example, to test $A2$, we will respectively select 1K samples from the testing sets of SMTP and Weibo. And then we randomly fulfill this new testing set



(a) Precision (b) Recall

Fig. 8. Precision and recall vs. #applications.

with traffic from other applications' testing set and the noise set, until the noise rate reaches 80%.

In BMAL process, we assume an initial labeled set with 20 positive flows in total for all interested applications, and select 100 samples each time from the unlabeled flows. Without otherwise specified, all content-based approaches are trained with 300 labeled flows along with all unlabeled flows (if needed). These labeled flows are selected from the BMAL process, and for REPLAY/BMAL, we randomly select 300 positive flows from the corresponding training set.

All experiments are conducted on an x86 platform (20× Intel 2.2Ghz CPU, 64GB memory) with 2×Nvidia Tesla K80 GPU (12GB memory for each GPU). Our implementation can use both GPUs at the same time for acceleration.

B. Performance

Accuracy. We verify the correctness of REPLAY by comparing the extraction results with the pre-generated labels. Two metrics are considered in the evaluation: precision and recall ("Pre" and "Rec" in the figure), which are defined as $\frac{TP}{TP+FP}$, and $\frac{TP}{TP+FN}$, respectively. Here TP denotes the number of interested bytes that are correctly extracted, FP denotes the number of meaningless bytes that are wrongly extracted, and FN denotes the number of interested bytes that are missed. Note that here we calculate the byte-level accuracy, and the accuracy after post-processing is shown in Section VI-E.

Fig. 8 shows the accuracy of all four content-based approaches trained with different number of applications. REPLAY yields a precision of 98.6% and a recall of 97% averagely over all cases. The accuracy is quite stable, proving the application-oblivious feature of REPLAY.

REPLAY/SSL achieves a good precision (∼93%), since it can correctly recognize the testing samples that are very similar to the training ones. However, its recall is unacceptable (lower than 80%), because hundreds of labeled samples are far less than it requires to completely understand the extraction. Recall REPLAY/BMAL is trained with pure positive samples that are sub-optimal, it therefore suffers a ∼3% accuracy penalty compared to REPLAY. We note that it is an ideal case of REPLAY/BMAL, since it could be expensive to obtain 300 positive samples without BMAL, and if we randomly select and label 300 samples, most of them would be negative, which will dramatically lower the accuracy. REPLAY/SP checks every byte of the input, and expects to be the most accurate approach, while REPLAY achieves a near-optimal accuracy with only ∼1% degrade than REPLAY/SP.

As a comparison, when parsing HTTP, QQ and DNS protocols, a specification-based parser results in 11.7% and 2.4% fault rate (the mis-identified bytes divided by the total bytes) with coarse- and fine-grained RegExes, respectively [13]. The major reason to its low accuracy is the high false-negative rate from the conservative specification writing, as mentioned in Section II-A. Another reference is that the SMTP specification generated by a reverse engineering approach (ProDecoder) achieves lower accuracy (95% precision), but is trained by much more samples (4,500 positive) [16].

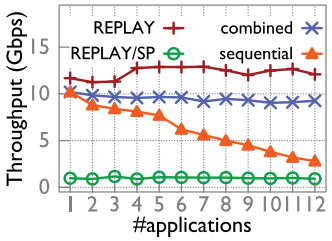


Fig. 9. Parsing speed.

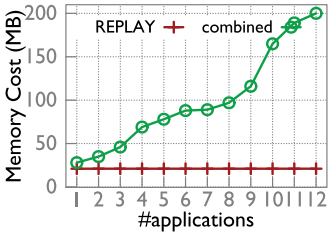


Fig. 10. Memory cost.

Parsing speed. We compare the parsing speed of REPLAY and REPLAY/SP to show the boost of the throughput due to the stop trigger. In addition, we also involve the two specification-based parsers in comparison: although they are executed on a single CPU core instead of GPU, we can observe their performance trends with the number of applications.

The results in Fig. 9 show that REPLAY can achieve stable throughput around 12Gbps with different number of applications involved. We observe higher throughput for applications $A4$–$A12$ because their extraction rate is small (see Table II), which means the stop trigger can skip more bytes.

REPLAY/SP is also scalable, but with a much lower throughput of ∼1Gbps due to the byte-by-byte checking.

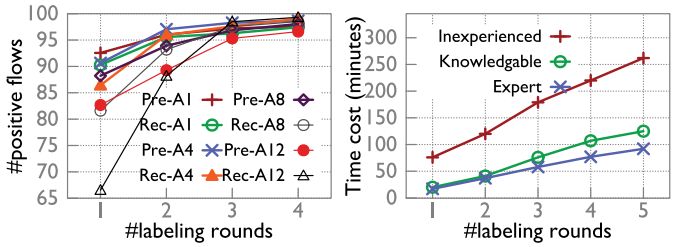
On the other hand, the combined parser can also achieve stable throughput, while the performance of sequential parser linearly degrades when the number of applications grows.

C. Overhead

Memory cost. We measure the memory cost of REPLAY and the combined parser. Note that REPLAY is accelerated by parallel parsing, so it will greedily copy its RNN for better performance. Here we only measure the off-line memory cost of a single parsing structure to observe the trend, *i.e.*, a single RNN vs. the combined automaton. Fig. 10 shows the memory cost trend with the increase of the number of applications. REPLAY consumes exactly the same memory (21MB) for all experiments, while the combined parser suffers a rapidly growing memory cost from 28MB to 200MB.

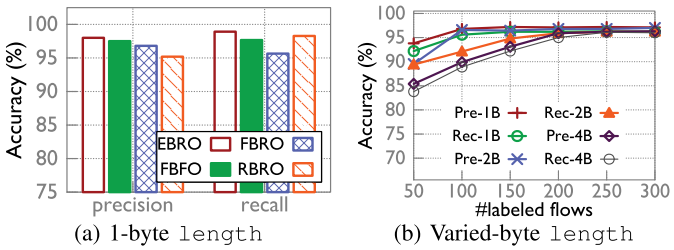
Labeling efforts. The major human efforts involved in REPLAY is to obtain enough high-quality labeled flows. We evaluate the efficiency of BMAL by measuring the accuracy with different BMAL selection rounds. Fig. 11a shows that a 3-round labeling process that obtains 300 labeled flows is sufficient for achieving high accuracy on $A12$.

The other metric we consider is the time cost for human labeling. Specifically, to imitate the operators with different technical background, we recruit 12 persons in this



(a) Accuracy (expert) (b) Accumulated time ($A12$)

Fig. 11. Labeling efforts.



(a) 1-byte `length` (b) Varied-byte `length`

Fig. 12. Accuracy for complex extractions (TLV).

experiment, including five undergraduate students who have bare knowledge of network protocols, five master students who major in computer networks but has not used any L7 parser, and two authors of this paper as the experts. The labeling process is to manually check the contents of the selected flows, and write the positions of interested bytes in a separate file for each flow. We measure the average time cost of the labeling process for each group of the operators.

According to Fig. 11b, the inexperienced operators spend more time (less than 3 hours for labeling 300 $A12$ samples) than the other two experienced groups, but a master student can achieve nearly the same efficiency compared to an expert (less than 1 hour for labeling 300 $A12$ samples). We also find that the operators are more efficient after the first two labeling rounds, as they gradually gain a grasp of the labeling patterns. In addition, we verify the label correctness, *i.e.*, whether the operator labels the correct field. Specifically, we compare the manually labeled results with the gold labels from the specifications, and observe that even the inexperienced operators can guarantee a high label correctness. We finally emphasize that the labeling tasks can be parallel conquered by multiple operators due to the independence of each label, as discussed in Section III-B.

Training time. The combination of SSL and BMAL conducts a two-stage training process: (1) unsupervised training on the complete data sets, and (2) iterative supervised training on the newly labeled flows.

For unsupervised training, we randomly select and inject 50 samples into RNN, and iterate this process for 50,000 times, which in total takes approximately 5 hours. We note that the unsupervised training is a one-time cost, unless the target network has been significantly changed.

For supervised training, we inject the same number of samples, and iterate until the loss converges. We measure the accumulated time costs for training, *e.g.*, for REPLAY, the cost includes the time of selecting samples with BMAL, the supervised training, and the stop trigger training, but excludes the time of manually labeling. The results report that REPLAY takes ∼60 minutes to train 300 labeled flows on $A12$.

D. Generalization

In above experiments, we train and evaluate REPLAY on datasets collected from/by the same source, which raises the generalization concerns: can REPLAY perform well, if (1) the target application is with more complex format, or (2) the testing dataset is different from the training set? In this section, we use two cases to tap the potential of REPLAY's generalization ability. Accuracy is the focus of these experiments.

Complex extractions. One concern of content-based parser is whether it can perform well for complex extraction behaviors. We choose the type-length-value (TLV) format as the building block to simulate a complex scenario, since to extract `value` requires an understanding of `length`, making it a complex context-sensitive grammar.

Specifically, we obtain four data sets, each of which contains 500 TLV samples. For each sample, we synthesize 100 TLV blocks following four principles: fixed blocks in fixed order (FBFO), fixed blocks in random order (FBRO), random blocks in random order (RBRO), and recursive blocks in random order (EBRO). Here a "random block" means a random `type`, while the other two fields are always generated randomly. The length of `type` and `length` is one byte. For EBRO, we generate 50 parent blocks, each of which contains a random child block as its `value`. We assume the interested content is the `value` of a specific `type`. For each data set, we use 100 samples as the labeled samples, and use the rest mixed with other data sets as the unlabeled samples.

As the results shown in Fig. 12a, REPLAY can still reach ∼97% for both precision and recall. This is because REPLAY learns a distribution of the context, which helps it localize the beginning and ending of `value` with varied `length`.

We further test REPLAY on RBRO with a varied-length `length`, ranging from 1 byte to 4 bytes. As shown in Fig. 12b, 100 samples for 4-byte `length` can only achieve ∼89% of precision and recall, because they are not sufficient to learn the abstract relationship between `length` and the factual length of `value`. But the accuracy is rapidly growing with more samples, and when training with 200 samples, the 4-byte case can also reach ∼95.5% precision and recall.

Testing with different datasets. It would largely improve the practicality of deploying REPLAY if the extraction model can perform well on a dataset that is different from the training dataset. Specifically, we note that the SMTP and MySQL traffic listed in Table I are collected by fuzzing the jmeter sampler [46] and random query generator [47], which is much more diverse than the SMTP and MySQL traffic from the real users. Therefore, we retrain the REPLAY with another SMTP and MySQL traces captured from the campus network, and test it with the fuzzing traffic, to evaluate whether REPLAY can perform well even some cases are not included in the training dataset. Fig. 13 compares the accuracy when testing on the same (real) and different (synthetic) dataset, and shows that the different testing sets lower 3.25% of the accuracy in average. This results suggest that, even in this unfavorable case, REPLAY can still provide a minimum guarantee when handling a broad range of network conditions.

E. Diving into REPLAY

How SSL improves the accuracy. As mentioned in Section IV-B, the unsupervised phase of SSL helps to

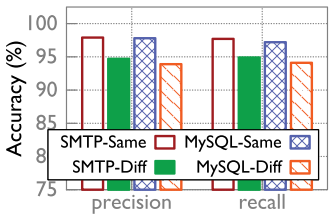


Fig. 13. Testing on different data sets.

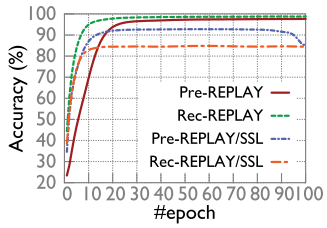


Fig. 14. Accuracy and convergence with SSL ($A1$).

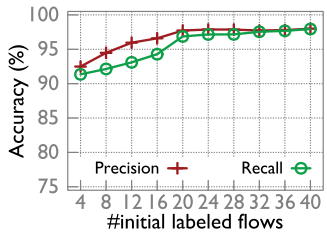


Fig. 15. Accuracy vs. #initial flows (300 samples, $A12$).

understand the inherent features of the traffic, which can improve the accuracy of the final model. To demonstrate this feature, we compare the accuracy trends of REPLAY and REPLAY/SSL with different training epochs. Fig. 14 shows that REPLAY can achieve much better accuracy than REPLAY/SSL, but converges a little slower. This is because REPLAY/SSL can easily converge to a model that perfectly classifies the limited labeled samples, though it may not be accurate in testing phase. While with SSL, the supervised phase has to involve the knowledge obtained from the unsupervised phase, which may take more time for convergence. Note that the precision of REPLAY/SSL gets lower after 90 epochs, because continuously training the minor samples (300) can result in the over-fitting issue. While thanks to the unsupervised phase, REPLAY will have an accurate model long before the over-fitting happens.

Trade-off of the ASCII-based embedding. As mentioned in Section V, the ASCII-based (binary) embedding trades off the accuracy with the parsing efficiency. We compare the accuracy and parsing speed with binary embedding and one-hot embedding in Table III, which reports that REPLAY with binary embedding is three times faster than it with the one-hot embedding, at the cost of only 1.8% degradation of accuracy.

Initial number of labeled flows. The initial labeled flows are used to train the first extraction model, and therefore can benefit next rounds of BMAL. Starting from different number of initial labeled flows, we perform BMAL process 3 rounds on $A12$ to obtain 300 samples, and measure the precision and recall. Fig. 15 shows that REPLAY can maintain its accuracy with only 20 initial labeled flows, which can be seen as a reasonable assumption of most L7 parsing scenarios.

Total number of labeled flows. Some may concern if REPLAY will require many more labeled flows when the number of applications grows. Fig. 16 shows the accuracy of

TABLE III
TRADE-OFF OF ASCII-BASED EMBEDDING ($A12$)

Embedding	Precision	Recall	Throughput
ASCII	97.2%	96.0%	12.1Gbps
One-hot	98.3%	98.5%	4.0Gbps

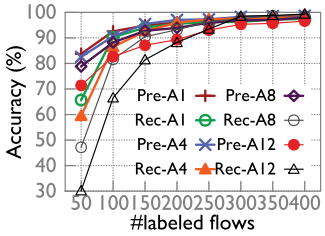


Fig. 16. Accuracy vs. #labeled flows (REPLAY).

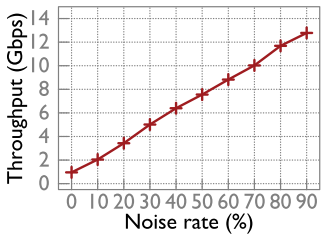


Fig. 17. Parsing speed vs. noise rate ($A1$).

REPLAY on four datasets, trained with different the number of labeled flows, where we have two observations. First, it indeed needs more labeled flows when involving more applications into the unified extraction model, *e.g.*, 200 samples for $A1$ can obtain ∼95% of accuracy, but can only reach ∼90% for $A12$. Second, the number of samples for each application actually decreases when the number of applications grows: to achieve 97% of accuracy, $A12$ requires 300 samples in total, *i.e.*, 15 samples per application, while $A1$ needs 250 samples to achieve similar accuracy for a single application.

The reason to this counter-intuitive fact is that for a certain application, the labels from other applications, *i.e.*, the true negative samples, also contribute to its accuracy. Specifically, the positive samples from application Y not only improves its own accuracy, but also helps the accuracy of application X by serving as the negative samples for X. As a result, we believe the efforts for involving more applications are scalable.

Parsing noise. The major throughput boost of REPLAY comes from the stop trigger, *i.e.*, the more data can be dropped, the better throughput can be realized. We test how this factor impacts the throughput of REPLAY by adjusting the noise rate in the test traffic when parsing $A1$. Note that $A1$ has an extract rate of ∼70%, so when noise rate is 0%, the actual uninteresting data is ∼30%.

Fig. 17 shows the parsing speed boost with the increase of noise rate (12.8Gbps on 80%-noise). We also emphasize that REPLAY can only achieve 0.97Gbps when most data is meaningful for the extraction. We believe this is a better trade-off compared with the application-dependent penalty, because applications can be added very frequently, but the noise rate is relatively high and stable in most scenarios.

Dropping rate. We measure the dropping rate of REPLAY to confirm the effect of the stop trigger. Here the dropping rate is defined as the factual dropped bytes divided by the uninterested bytes. Fig. 18 shows that REPLAY drops ∼80% bytes for all cases. However, the noise rate is 80%, while the factual rate of the uninterested bytes should be much higher,

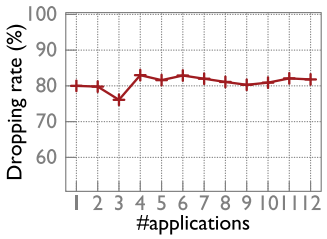


Fig. 18. Dropping rate vs. #applications (REPLAY).

since the extract rates of many datasets are very low. For example, $A1$ has an extract rate of 70%, so the total uninterested bytes for the testing data set should occupy ∼86%, *i.e.*, $NoiseRate + (1 - NoiseRate) \times (1 - ExtractRate)$, while for $A12$, the uninterested bytes could occupy ∼99%.

The reason is that the stop trigger can only tailor the rest of the input, which means if the interested bytes are located in the end of input, REPLAY can barely drop the text. On the other hand, the pure noise can be dropped with a glance at the first few bytes. Therefore, REPLAY can efficiently drop the noise ($NoiseRate$, *i.e.*, 80%), but is not stable for dropping the other uninterested bytes ($1 - ExtractRate$). This finally results in a dropping rate that is slightly higher than the noise rate. We discuss this limitation of REPLAY in Section VII.

Accuracy with extraction post-processing. As mentioned in Section V, we use a simple heuristic to compose a continuous extraction result for each type of extraction behavior. We expect the post-processed results can still maintain the high accuracy. The reason is that REPLAY can achieve high accuracy in byte-level ($> 97\%$). That is, it is quite unlikely that REPLAY generates two or more continuous false positives or negatives. As a result, the longest continuous results should be correct, the long-distant (wild) extraction results should be false positives, and the short gaps between two continuous results are likely to be false negatives.

We set the distance threshold very aggressively, *i.e.*, 2 bytes, and verify the accuracy before and after applying the extraction post-processing on $A12$. The result reports that REPLAY can achieve an even higher accuracy (0.1% of precision and 0.4% of recall), because some simple false positives and negatives can be fixed by the heuristics.

VII. Discussion and Limitations

Unreadable and encrypted protocols. REPLAY can learn and parse both text- and binary-based L7 protocols. The challenge of handling the latter one is that the operator may not achieve high labeling effectiveness if the content is unreadable. For the similar reason, REPLAY may perform poorly on encrypted protocols, because the operator cannot provide effective labels on the encrypted content. However, as we discussed in Section I, many valuable L7 data are unencrypted and readable, ensuring the significance of REPLAY.

Overlapped extractions. Using H-softmax as output layers means that REPLAY can only output one extraction behavior for each byte, which is a limitation of REPLAY. Moreover, unless the two applications label the same samples, such overlaps cannot be identified, because without protocol knowledge, the parser can only distinguish the extraction by the content. We will explore this valuable question in our future work. Fortunately, such overlapping labels could be rare cases in practice, since the operators using REPLAY usually train their own models for specific networks. That is, REPLAY is not meant to serve as a single large parser for everyone, but a customized one trained with specific samples, which can be easily managed by each operator of different networks. Finally, even the operators work independently on the same parser, it is still easy for them to feed the samples to REPLAY before labeling them, so that they can verify if REPLAY can already extract the interested content. If so, they can directly reuse the parsing result instead training with different labels.

Skipping more bytes. As discussed in Section VI-E, the stop trigger can fast drop the noise traffic, but cannot efficiently save the computation inside an interested flow. This limitation results from that the stop trigger can only learn the "end" of the interested content, but cannot be aware of the uninterested content in-between two interested pieces. It is possible to design a new trigger, say "skip trigger", that can identify whether a batch of input bytes contains any interested content, and directly drop them if the answer is no. We leave this design for our future study.

Privacy issue. Like all DPI tools, REPLAY also faces the problem that how to prevent from being used for nefarious purposes. Apart from the common defenses that designing a DPI tool does not touch the legal part, we have the following specific arguments for REPLAY. (1) As discussed above, the encrypted content is still secured with REPLAY, because the operators cannot provide effective labels. (2) For the non-encrypted content, the key of protecting privacy is to prevent the unauthorized access. Otherwise, even without REPLAY, the larger human-efforts of building a parser are acceptable to the hackers, as long as the privacy is worthy enough.

VIII. Related Work

Specification-based parsers. All previous L7 parsers are based on specifications. Binpac [5], GAPA [22], Ultrapac [1] and FlowSifter [2] automatically generate parsers by the specifications written in their declarative languages. COPY [13] designs a distinguishable automaton for parsing multiple protocols simultaneously without speed or accuracy loss. These parsers can yield high accuracy, but at the risk of massive human labor and the performance penalty.

Reverse engineering-based approaches. This kind of approaches distills message patterns hidden in the traffic. Polyglot can extract message formats by analyzing application binaries [14], which are not easy to fetch in the network. Several approaches adopt the machine learning technique to automate the protocol inference [15], [16], [48], while all those solutions require considerable human intervention to prepare training data for each protocol. Besides, they only focus on the target protocol but ignore the potential similar ones, raising lots of false positives if the application is carried by a common embedded protocol like JSON and XML.

Deep learning on text. NN has shown great success when handling natural language problems, such as word embedding [38], speech recognition [25], machine translation [24], *etc*. RNN is especially effective to interpret complex texts from structured data, *e.g.*, XML/JSON files [49]. Hence, we refer RNN as a good start of realizing a content-based parser, but directly applying classic RNN will result in incomplete context awareness of the whole text, large labeling efforts, and low parsing speed. Snorkel [50] uses the weak supervision to fast obtain the labeled data, which shares the goal of reducing the human labeling efforts. However, Snorkel can

only label the whole samples instead of the certain texts, which means it can only serve as a flow classifier, instead of an L7 parser that extracts underlying fields. REPLAY addresses the above problems, and achieves content-based parsing without any application dependence, targeting at line-rate network applications.

IX. Conclusion

REPLAY shifts the focus of L7 parsing from the specifications to the contents with enhanced deep learning techniques. Operators can just care about "what" content to extract, and leave the question of "how" to extract the content to REPLAY, minimizing the human interventions of handling new applications. In real-time parsing, REPLAY yields high accuracy and throughput, which is stable and oblivious to the applications. To the best of our knowledge, REPLAY is the first approach to model and realize L7 field extraction without specifications, which enables effective, efficient and scalable L7 parsing for fine-grained and domain-specific scenarios.

References

[1] Z. Li *et al.*, "Netshield: Massive semantics-based vulnerability signature matching for high-speed networks," in *Proc. ACM SIGCOMM*, 2010, pp. 279–290.

[2] C. Meiners, E. Norige, A. X. Liu, and E. Torng, "FlowSifter: A counting automata approach to layer 7 field extraction for deep flow inspection," in *Proc. IEEE INFOCOM*, Mar. 2012, pp. 1746–1754.

[3] (2018). *Mcafee Data Loss Prevention Prevent*. [Online]. Available: https://bit.ly/2JnpZP3

[4] (2018). *Symantec Data Loss Prevention*. [Online]. Available: https://bit.ly/3468MDv

[5] R. Pang, V. Paxson, R. Sommer, and L. Peterson, "binpac: A yacc for writing application protocol parsers," in *Proc. 6th ACM SIGCOMM Conf. Internet Meas.*, 2006, pp. 289–300.

[6] C. Hu, H. Li, Y. Jiang, Y. Cheng, and P. Heegaard, "Deep semantics inspection over big network data at wire speed," *IEEE Netw.*, vol. 30, no. 1, pp. 18–23, Jan. 2016.

[7] (2017). Benefits of Layer 7 Load Balancing. [Online]. Available: https://bit.ly/2UpJiNU

[8] (2017). *NFV Monitoring*. [Online]. Available: http://qosmos.com/telecoms/nfv-monitoring/

[9] A. Khurshid, X. Zou, W. Zhou, M. Caesar, and P. B. Godfrey, "Veriflow: Verifying network-wide invariants in real time," in *Proc. USENIX NSDI*, 2013, pp. 15–27.

[10] (2018). *Huawei Esight Application Manager*. [Online]. Available: https://bit.ly/33WFTJJ

[11] (2017). *Netware Core Protocol*. [Online]. Available: https://bit.ly/2UpIOY6

[12] (2017). *Protocol and Metadata Support*. [Online]. Available: https://bit.ly/33V0XjB

[13] H. Li, C. Hu, J. Hong, X. Chen, and Y. Jiang, "Parsing application layer protocol with commodity hardware for SDN," in *Proc. ACM/IEEE Symp. Architectures for Netw. Commun. Syst. (ANCS)*, May 2015, pp. 51–61.

[14] J. Caballero, H. Yin, Z. Liang, and D. Song, "Polyglot: Automatic extraction of protocol message format using dynamic binary analysis," in *Proc. ACM CCS*, 2007, pp. 317–329.

[15] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker, "Unexpected means of protocol inference," in *Proc. 6th ACM SIGCOMM Internet Meas. (IMC)*, 2006, pp. 313–326.

[16] Y. Wang *et al.*, "A semantics aware approach to automated reverse engineering unknown protocols," in *Proc. 20th IEEE Int. Conf. Netw. Protocols (ICNP)*, Oct. 2012, pp. 1–10.

[17] R. Smith, C. Estan, and S. Jha, "XFA: Faster signature matching with extended automata," in *Proc. IEEE Symp. Secur. Privacy*, May 2008, pp. 187–201.

[18] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," in *Proc. Conf. Appl., Technol., Architectures, Protocols Comput. Commun. (SIGCOMM)*, 2006, pp. 339–350.

[19] (2018). *Simple Mail Transfer Protocol*. [Online]. Available: https://tools.ietf.org/html/rfc5321

[20] (2008). *SMTP Commands Reference*. [Online]. Available: https://bit.ly/2UO9gcO

[21] C. Shen and L. Huang, "On detection accuracy of L7-filter and OpenDPI," in *Proc. 3rd Int. Conf. Netw. Distrib. Comput.*, Oct. 2012, pp. 119–123.

[22] N. Borisov, D. Brumley, H. J. Wang, J. Dunagan, P. Joshi, and C. Guo, "Generic application-level protocol analyzer and its language," in *Proc. NDSS*, 2007, pp. 1–13.

[23] (2018). *Metathink: Big Data or Challenging Data?*. [Online]. Available: https://bit.ly/2WPGJq9

[24] Y. Wu *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," 2016, *arXiv:1609.08144*. [Online]. Available: http://arxiv.org/abs/1609.08144

[25] A. L. Maas, Q. V. Le, T. M. O'Neil, O. Vinyals, P. Nguyen, and A. Y. Ng, "Recurrent neural networks for noise reduction in robust ASR," in *Proc. Interspeech*, 2012, pp. 22–25.

[26] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proc. EMNLP*, 2014, pp. 1724–1734.

[27] F. Morin and Y. Bengio, "Hierarchical probabilistic neural network language model," in *Proc. Aistats*, vol. 5, 2005, pp. 246–252.

[28] R. Longadge and S. Dongre, "Class imbalance problem in data mining review," 2013, *arXiv:1305.1707*. [Online]. Available: http://arxiv.org/abs/1305.1707

[29] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: http://arxiv.org/abs/1412.6980

[30] M. C. Mozer, "Induction of multiscale temporal structure," in *Proc. Adv. Neural Inf. Process. Syst.*, 1992, pp. 275–282.

[31] C. Xiong, S. Merity, and R. Socher, "Dynamic memory networks for visual and textual question answering," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 2397–2406.

[32] G. Zhou, Y. Zhou, T. He, and W. Wu, "Learning semantic representation with neural networks for community question answering retrieval," *Knowl.-Based Syst.*, vol. 93, pp. 75–83, Feb. 2016.

[33] C.-Y. Liou, W.-C. Cheng, J.-W. Liou, and D.-R. Liou, "Autoencoder for words," *Neurocomputing*, vol. 139, pp. 84–96, Sep. 2014.

[34] T. Mikolov, M. Karafiát, L. Burget, J. Černocký, and S. Khudanpur, "Recurrent neural network based language model," in *Proc. Interspeech*, vol. 2, 2010, p. 3.

[35] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, Nov. 1997.

[36] S. C. H. Hoi, R. Jin, and M. R. Lyu, "Large-scale text categorization by batch mode active learning," in *Proc. 15th Int. Conf. World Wide Web (WWW)*, 2006, pp. 633–642.

[37] S. Chakraborty, V. Balasubramanian, Q. Sun, S. Panchanathan, and J. Ye, "Active batch selection via convex relaxations with guaranteed solution bounds," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 10, pp. 1945–1958, Oct. 2015.

[38] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 2013, pp. 3111–3119.

[39] (2018). *Tensorflow*. [Online]. Available: https://www.tensorflow.org

[40] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[41] V. Pham, T. Bluche, C. Kermorvant, and J. Louradour, "Dropout improves recurrent neural networks for handwriting recognition," in *Proc. 14th Int. Conf. Frontiers Handwriting Recognit.*, Sep. 2014, pp. 285–290.

[42] (2018). *Imdb*. [Online]. Available: http://www.imdb.com/

[43] (2018). *Asahi Shimbun*. [Online]. Available: http://www.asahi.com/ajw/

[44] (2016). *XML Sample Generator*. [Online]. Available: https://bit.ly/2UrrMJ5

[45] (2018). *Soapui*. [Online]. Available: https://www.soapui.org/

[46] (2017). *Apache Jmeter*. [Online]. Available: http://jmeter.apache.org/

[47] (2008). *Random Query Generator*. [Online]. Available: https://launchpad.net/randgen

[48] X. Yun, Y. Wang, Y. Zhang, and Y. Zhou, "A semantics-aware approach to the automated network protocol identification," *IEEE/ACM Trans. Netw.*, vol. 24, no. 1, pp. 583–595, Feb. 2016.

[49] R. Jozefowicz, W. Zaremba, and I. Sutskever, "An empirical exploration of recurrent network architectures," in *Proc. ICML*, 2015, pp. 1–9.

[50] (2020). *Snorkel*. [Online]. Available: https://www.snorkel.org

Hao Li received the Ph.D. degree in computer science from Xi'an Jiaotong University in 2016. He is currently an Assistant Professor with School of Computer Science and Technology, Xi'an Jiaotong University. His main research interests include SDN/NFV and network measurement.

Zhengda Bian received the B.S. and M.S. degrees in computer science from Xi'an Jiaotong University in 2015 and 2018, respectively. He is currently pursuing the Ph.D. degree with the National University of Singapore. His current research interests include performance analysis and design of resource management and scheduling systems for large-scale distributed deep learning in GPU clusters.

Peng Zhang (Member, IEEE) received the Ph.D. degree in computer science from Tsinghua University in 2013. He is currently an Associate Professor with the School of Computer Science and Technology, Xi'an Jiaotong University. His research interests include verification, measurement, privacy, and security in computer networks.

Zhun Sun received the Ph.D. degree in information science from Tohoku University in 2018. She has been working with the RIKEN Center for Advanced Intelligence Project as a Post-Doctoral Researcher until 2020. She is currently working with BIGO Inc., as a Research Scientist. Her main research topics include computer vision with deep learning, generative model, computational graphics, and other deep neural networks related tasks.

Chengchen Hu (Member, IEEE) received the Ph.D. degree in computer science from Tsinghua University in 2008. In 2017, he was a Professor and the Head of the Department of Computer Science and Technology, Xi'an Jiaotong University. He is currently the Founding Director of Xilinx Labs Asia Pacific, Singapore, and also leading research on networked processing systems. His research interests include networked systems and programmable networks.

Qiang Fu (Member, IEEE) received the Ph.D. degree in telecommunications engineering from The University of Queensland, Australia. He is currently a Senior Lecturer in network/software engineering with the School of Engineering and Computer Science, Victoria University of Wellington, New Zealand. His research interests include Internet architecture and protocols, wireless and mobile systems, and network measurement and security.

Tian Pan received the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, in 2015. He is currently an Assistant Professor with the Beijing University of Posts and Telecommunications. His research interests include SDN, programmable data plane, and satellite networks.

Jia Lv received the Ph.D. degree in control science and engineering from Xi'an Jiatong University. He is currently a Researcher with the Huawei Noah's Ark Lab. His research interests mainly focus on intelligent operation and maintenance of telecommunication network, including abnormal detection and telecom network diagnosis.